Search with Wildcards on a Quantum Computer Project Description

Liam Adams and Alec Landow

October 8, 2020

https://pages.github.ncsu.edu/lbadams2/qc_final_proj/

1 The Problem

We will implement a "Search with Wildcards" algorithm on IBM Qiskit, and run the implementation with small input sizes both on a simulator and on an IBM quantum computer. We will then optimize the algorithm to improve the success rate on an IBM quantum computer.

2 Selecting an Algorithm

The first task is to select the algorithm. When searching for an algorithm to implement, we considered the following factors.

- Is the algorithm interesting.
- Can it be done in a reasonable amount of time.
- Is there a reasonable amount of documentation and other references to help guide us.

We reviewed most of the papers in the References section ([1] [2] [3] [4] [5] [6]), and outlined two in detail below.

1. *Quantum algorithms for search with wildcards and combinatorial group testing* [5]
   The search with wildcards problem consists of taking an n-bit string x as input and determining the full string x with the minimum number of queries.

   The algorithm runs by building the string x gradually, increasing the number of correct bits at each step. Specifically [5] proves that for any $k = n - O(\sqrt{n})$, there is a quantum measurement on input $|\psi_x^k\rangle$ which outputs $\bar{x}$ such that the expected Hamming distance $d(x, \bar{x})$ is $O(1)$.

   The circuit begins by creating a superposition of k inputs (each input size k or k inputs?), with each input a subset of of the unknown string x. Now the task is to find the remaining $O(\sqrt{n})$ bits. Each input represents a possible subset of x, the sum of

them exceeds n but we are only interested in the one correct subset. Now
1) Pass the inputs through a unitary transformation U to map the input state $|\psi_x^k\rangle$ to a state $H_o \otimes H_g$ in which H_o is the output register and H_g is the rest of the state.
2) Measure H_o, which is a guess $\bar{x}$ for the string x we are trying to determine.

Now define a sequence of numbers $n_0, ..., n_l$ with $n_l = n$. The algorithm consists of l stages, terminating at the final stage when the guess is n bits. First create $|\psi_x^{n_0}\rangle$ by creating superposition of inputs of size n_0. Then measure the bits of the inputs using the 2 step procedure described above.

The subsequent stages receive $|\psi_x^{n_{s-1}}\rangle$ as input and output $|\psi_x^{n_s}\rangle$. To do this first transform $|\psi_x^{n_{s-1}}\rangle$ to

$$\sum_{S\subset[n],|S|=n_s} |S\rangle\,|\psi_{x_s}^{n_{s-1}}\rangle$$

. In this expression S us a subset of the n-bit string x. Then apply the unitary transformation to the registering storing $|\psi_{x_s}^{n_{s-1}}\rangle$ which produces a measurable state in H_o. Measure H_o to get $\bar{x_s}$, and verify if it is equal to x_s. If they are equal then we have the state

$$\sum_{S\subset[n],|S|=n_s} |S\rangle\,|x_s\rangle\,|\psi_s\rangle$$

where $|\psi_s\rangle$ is the state in the H_g register referenced previously. Now apply the transformation $|S\rangle\,|\psi_s\rangle \to |S\rangle\,|0\rangle$ and discard H_g.

If the $\bar{x_s}$ does not equal x, use a binary search to find one bit differing between the 2 strings and flip that bit in $\bar{x_s}$. Now query again to see if the two are equal. If they still aren't equal repeat the binary search, if they are equal we have the state in equation 2 and proceed as before.

If we choose $l = O(\sqrt{n})$ then the complexity of the algorithm is $O(\sqrt{n}log(n))$ due to the binary search when the guess is incorrect.

2. *Quantum pattern matching* [1]

The algorithm that Mateus and Omar propose is based on the Grover search algorithm. The Mateus and Omar algorithm provides for matching a pattern to an unsorted string input, and returns the position of the closest substring to the pattern with what the authors describe as "non-negligible" probability in $\mathcal{O}(\sqrt{n})$ queries. Instead of creating a query function that will only work to find one item, the authors propose a method of creating query functions that can be reused to query multiple different items.

There is a query function for each symbol in the alphabet. The trade-off with this approach is reducing run-time while increasing compile-time complexity.

Each position in the string is indicated by a state $|i\rangle$. The initial state $|\psi_0\rangle$ is an entanglement of all states for which the first character of the pattern p, p_1, is followed by the second character in the pattern, p_2. The search is done by applying the query operator Q_σ for each symbol σ of the alphabet Σ:

$$Q_\sigma(|i\rangle \otimes |b\rangle) = |i\rangle \otimes |f_\sigma \oplus b\rangle \tag{1}$$

where

$$\begin{aligned}&|b\rangle \text{ is an auxiliary qubit}\\ &\text{and } f_\sigma = \begin{cases} 1 & \text{if the } i\text{th letter of the input string is } \sigma \\ 0 & \text{otherwise}\end{cases}\end{aligned} \tag{2}$$

The main idea is to use the query function to repeatedly shift the phase of the target symbol. This is done by applying the query operator for a random symbol in the pattern to what the paper describes as the "corresponding position", which is not immediately clear and warrants further thought. It appears that amplitude amplification is then applied to the state that has been phase-shifted the most. The process is then repeated for the first $\sqrt{N-M+1}$ positions in the input string, where N is the number of characters in the input string, and M is the number of characters in the pattern.

The paper gives the circuit diagram for constructing the initial state given an input string of $N = 2$. Each query operator can be constructed by implementing a permutation operator. Specifically, the authors detail using Gray codes [7] implemented using a series of CNOT gates.

3 Implementation

If we choose to implement the reusable pattern matching circuit of [1], implementation will consist of constructing the query operators for each symbol of the alphabet, constructing the initial state, and then figuring out how to apply the gates of the alphabet appropriately to an input string given an input pattern.

The Qiskit tutorial on Grover's Algorithm [8] will also be a good reference.

4 Optimization

At this point we do not believe we are familiar enough yet with the basic implementation of the algorithm to consider optimization. However optimization may include reuse of ancilla qubits or dividing large inputs into separate queries to meet the constraint of a limited number of qubits.

5 Timeline

- Thursday, October 8, 2020: Project Description Complete
- Thursday, October 15, 2020: Draw in Qiskit a diagram of several symbol circuits, as well as a simple complete, working circuit
- Sunday, October 25, 2020: Partial implementation complete
  - Construction of the symbol gates complete
  - Construction of the input state complete
- Sunday, November 1, 2020: Full pattern matching program works for arbitrary number of qubits
- Sunday, November 8, 2020: Pattern matching program is optimized for use on a real quantum computer
- Tuesday, November 17, 2020: Project Complete

References

[1] P. Mateus and Y. Omar. Quantum pattern matching, 2005.

[2] Vikram Menon and Ayan Chattopadhyay. Quantum pattern matching oracle construction, 2020.

[3] Ashley Montanaro. Quantum pattern matching fast on average, 2015.

[4] H. Ramesh and V. Vinay. String matching in $\mathcal{O}(\sqrt{n} + \sqrt{m})$ quantum time, 2000.

[5] Andris Ambainis and Ashley Montanaro. Quantum algorithms for search with wildcards and combinatorial group testing, 2013.

[6] Kapil Kumar Soni and Akhtar Rasool. Pattern matching: A quantum oriented approach, 2020.

[7] Gray code. https://en.wikipedia.org/wiki/Gray_code.

[8] Qiskit tutorial on grover's algorithm. https://qiskit.org/textbook/ch-algorithms/grover.html.